*In this video I'm going to give a quick taste of scala, which will hopefully whet your appetite for learning more about the language*

*Scala has a reputation for having a steep learning curve and this is partly true. It's not too hard to pick up the basics but to truly master the language takes a fair bit of patience. The good news is that you can be productive in Scala in a fairly short amount of time.*

*So let's get started with some examples*

***REPL***

*Scala comes bundled with an interactive console called the repl. That’s REPL, which stands for Read-Eval-Print-Loop.*

*To invoke the repl, fire up a terminal and type scala*

[New Terminal]

scala

*now you can type some expressions like*

1+2

*you can define a function that adds 2 to a number like this*

def plus2(x:Int) = x + 2

*and call the function*

plus2(5)

*See the res2? The repl automatically generates variables for each result, so if I want to reuse a result, I can just use the variable name. For example if I type res2, the value will be displayed.*

*Or I can use it in another expression like*

res2 + 4

**Literals**

*The syntax of Scala makes it easy to build data structures during development and testing. For example, suppose I want a map from words to lists of indices in a text document. I could type something like*

val m = Map("quick" -> List(10,23,46), "fox" -> List(4,16))

m: scala.collection.immutable.Map[String,List[Int]] = Map(quick -> List(10, 23), fox -> List(4, 16))

*To get the list of indices, I can use a function-like syntax to retrieve values for a key*

scala> m("quick")

*And of course I can compose functions to perform operations on the list that is returned, like I can get the head of the list*

scala> m("quick").head

*or the tail of the list*

scala> m("quick").tail

res5: List[Int] = List(23)

**Strongly Typed**

*So far Scala looks similar to scripting languages such as Groovy or Python. What’s the difference?*

*Well, let’s try an example in Python.*

python

>>> def f(x):

... y + 2

...

>>> f(3)

*We get an error when we call the function, but not when we define the function. So we might not discover this error until the code was deployed in production.*

*Now let’s try this in Scala*

> def f(x:Int) = y + 2

*Scala doesn’t let us define the function. It compiles the function and discovers an undefined variable.*

*Let’s look at another example in Python.*

>>> def f(x):

... x - "y"

...

f(x)

*Again, we get an error at runtime. Python is loosely-typed, which makes it easy to write code but leads to bugs.*

Now let’s try this in Scala

scala> def f(x:Int) = x - "y"

*Scala doesn’t allow such nonsense. It’s strongly typed and will catch type errors at compile time.*

**Quick sort**

*Let’s look at a meatier example of some code written in Scala.*

(swipe to full screen IntelliJ with QuicksortScalaShort)

*Here’s an implementation of quicksort in Scala which is 15 lines long including comments and whitepace or about 8 lines long if you’re just counting lines of code.*

*As a refresher on the quicksort algorithm,*

* *The algorithm takes an array as input and randomly selects an item from the array to be the partition value.*
* *Usually the first element of the array is picked for the partition.*
* *Then the array is partitioned into two subarrays- one with values less than the partition and one with values greater than the partition.*
* *The algorithm recurses on each of the subarrays and once the recursive calls complete the array is sorted.*

*So this code might look a little weird if you’re not familiar with Scala.*

*To get a better understanding of the code let’s experiment in the Scala console:*

*Let’s start with Line 8, which says object QuicksortScalaShort. You can think of an object as a class with static members. Another way of thinking of objects is as a singleton. Here’s an example:*

object Foo { val bar = "baz" }

*We’re defining an object named Foo with a single member variable called bar with value “baz”. We can access the member without doing any instantiation.*

Foo.bar

*On line 9, We have the function signature of the quicksort method. Quicksort takes as input an array of Ints and outputs an Array of Ints. Let’s define a simple function which modifies it’s first element.*

def f(a: Array[Int]):Array[Int] = { a(0) = 5; a }

f(Array(3,2))

*Back to line 9. A match operator is being applied to the input array. You can think of the match operator as a fancy switch statement. There are a series of cases that attempt to match the input and evaluate an expression if the match is successful. Let’s try some experiments to see match in action.*

*Let’s match an empty array against a case which expects an empty Array.*

Array() match { case Array() => "ok" }

*The input has a successful match so the expression returns ok*

*Now let’s try matching a non-empty array against a case which expects an empty array.*

Array(1,2,3) match { case Array() => “ok" }

*In this case we get an error since there’s no cases which match the input.*

*Now let’s try matching a two-element array against a case which expects an array with 2 elements.*

Array(1,2) match { case Array(one, two) => s"one is $one, two is $two" }

*Now let’s try matching a three-element array against a case which expects an array with 2 elements.*

Array(1,2,3) match { case Array(one, two) => s"one is $one, two is $two" }

*And surprise, we get a match error.*

*Now let’s try matching a three-element array against a case which expects a head element and zero or more tail elements. The syntax is like this:*

Array(1,2,3) match { case Array(head, tail@\_\*) => s"head is $head, tail is $tail" }

*So apparently this matcher grabs the remainder of the Array and creates a Vector, which is a resizable data structure. To convert the Vector to an Array, use the toArray operator*

Vector(2,3).toArray

*So now we can read lines 9 through 11*. *Line 9 is matching the input array. Line 10 is matching the case where the input array is empty, and line 10 is matching the case where the input array is non-empty and it splits it into a head and a tail.*

*Let’s look at line 12. Here, we’re applying the partition operator to the tail. The partition operator uses a boolean-valued function (called a predicate) to make a decision about which partition an element belongs to. The expression between the curly brackets is a function literal. Let’s play with a function literal.*

val f: Int => Boolean = { \_ < 5 }

*Here we’re defining a variable f whose type is a function which takes an Int as input and produces a Boolean output. This function compares returns true if the input is less than 5. The underscore is an anonymous variable, which saves us the trouble of thinking of a variable name.*

*Now we can use our function literal like a named function.*

f(4)

*Returns true*

f(5)

*Returns false*

*Now let’s try the partition function.*

Array(1,3,5,7) partition { \_ < 5 }

*The partition function creates a tuple of 2 arrays. The element in the tuple has elements which satisfy the predicate and the tuple element has elements which don’t satisfy the predicate.*

*Now let’s do the assignment:*

val (before, after) Array(3,5,7) { \_ < 5 }

*The result of the partition is matched against the structure of the lefthand side of the assignment. So the before variable gets the first element of the tuple and the after variable gets the second element of the tuple.*

before

after

Array (1,2) ++ Array(3) ++ Array(4,5)

concatenates arrays

*So in summary:*

*line 8 defines the object that encapsulates our quicksort function*

*line 9 defines the method signature and matches the input array against 2 cases*

*in line 10, if the input array is empty, we’re done, return the empty array*

*line 11 matches non empty arrays and splits the array into a head and a tail*

*line 12 converts the tail to an array, and partitions into 2 arrays based on the value of the head*

*line 13 recurses on the 2 subarrays and concatenates the results*

*So let’s try it out:*

QuicksortScalaShort.quicksort(Array(1,7,5,3))

*It works!*

*How many lines would it take in Java?*

*Here’s the code in Java - its about 50 lines*

*Is this a fair comparison? That’s debatable, since the partition function is provided in a Scala collections library*

*This does point out the power of the collections library. In Scala, there’s tons of useful operations you can do on collections, but that’s another topic*

*By the way, you can write code in the Java style in Scala, in which case the line count will be similar.*

*Here’s the algorithm written in Scala using a non-functional style.*

*You can see that the line count in the Scala version is slightly higher due to the use of nested functions. The benefit of using nested functions is that fewer parameters need to be passed around gratuitously.*

*Claims on savings in the amount of code vary. I would say in general you can save anywhere from 30 to 50% of the amount of code you’d write in Java.*

*Here’s one unscientific study. I downloaded the LinkedIn Play sample app and compared the size of demos that were written in both Scala and Java.*

[Show graph comparing play demos]

*Just as important as the quantitative savings in the number of lines of code is the qualitative shift in your thinking to a higher level of abstraction*

*For example, instead of thinking about looping over an array and performing operations on each item, you will think instead in terms of*

*transforming a collection of objects. So you’re thinking more about the problem you’re solving and less about the implementation, which leads to better productivity as a programmer.*

**JVM**

*There’s one final thing I want to show for this Scala intro.*

*Scala is Built on Jvm and can use Java*

*for example we can type*

new java.util.Date()

QuicksortJava.quicksort(Array(1,7,5,3))

**Summary**

*So those are few examples to give you a feel for what Scala’s about. We’re just scratching the surface, and in future videos we’ll learn a lot more about what Scala has to offer.*

*It gives you the conciseness of a scripting language with the safety of an industrial strength language*

*Hopefully that was informative and thanks for watching*